avtor Tine Lesjak, 63030315
mentor doc. dr. Branko Šter
predmet celularne strukture in sistemi
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
Ljubljana, 31. 05. 2007

Celularni programski algoritem - Polnjenje pravokotnika

Opis problema

Cilj naloge je s pomočjo celularnega programskega algoritma (CPA) razviti nabor pravil, ki v celularnem avtomatu (CA) napolnijo prostor pravokotne oblike omejen s štirimi stranicami - pravokotnik. Pri tem ostane prostor zunaj pravokotnika prazen.

Algoritem je razvit iz CPA-ja dr. Mosheja Sipperja. Njegov opis algoritma in knjigo *Evolution of Parallel Cellular Machines* je moč dobiti na spletni strani http://www.moshesipper.com/.

Specifikacije

Lastnosti celularnega avtomata (prostora) in algoritma:

- CA je dvodimenzionalen.
- CA je 2-stanjski. Stanje celice je lahko 1 (true) ali 0 (false).
- CA je neuniformen. Vsaka celica ima svoje pravilo.
- Sosednost je 5 (von Neumannova sosednost). Sosedne celice so poleg sebe zgoraj, desno, spodaj in levo.
- Začetna pravila so naključna.
- Vsak pravokotnik (ne cel CA) je napolnjen z enicami in ničlami naključno tako, da je zagotovljena večja variabilnost: najprej se naključno določi razmerje enic in ničel, nato se glede na razmerje naključno določijo stanja celic.
- Pravila so 32 bitna (2^5).
- CA ni cikličen. Celice, ki padejo izven prostora, imajo stanje 0. Fitnes celic izven prostora je 0. Pravila izven prostora imajo kodo 0.
- Stranice oz. robovi pravokotnika so ob vsaki začetni konfiguraciji nastavljeni naključno tako, da ne dosežejo roba prostora in da ne presežejo polovice prostora. Na primeru CA velikosti 8 x 8 celic (1-8):
    - zgornji in levi rob: 2, 3 ali 4,
    - spodnji in desni rob: 5, 6 ali 7.
- Spremembe stanj sosednjih celic so opisane po bitih pravila po vrsti: sredinska (ta), zgornja, desna, spodnja in leva.

Nastavljive lastnosti, na katerih sem izvajal meritve:

- CA je velik 8 x 8 celic. Ima 64 celic.
- Število izvajanj posamezne začetne konfiguracije (M) je 12.
- Število začetnih konfiguracij (C) je 200.
- Število generacij (G) je odvisno od meritve. Uporabil sem vrednosti 500 in 5000.
- Verjetnost mutacije pravila pri evoluciji je 0,001. Mutacija se zgodi le, če sta več kot dve sosednji pravili boljši (imata večji fitnes).

- Pri zadnji meritvi sem ob začetni konfiguraciji dodatno nastavil stanja celic v vseh štirih kotih pravokotnika na 1.

Algoritem sem napisal v programskem jeziku Java (različice 1.4.2). Razvijal sem ga v razvojnem okolju Eclipse.

Meritve

Meritve sem opravljal na svojem delovnem računalniku, ki ima 1500 MHz procesor AMD in 1 GB pomnilnika.

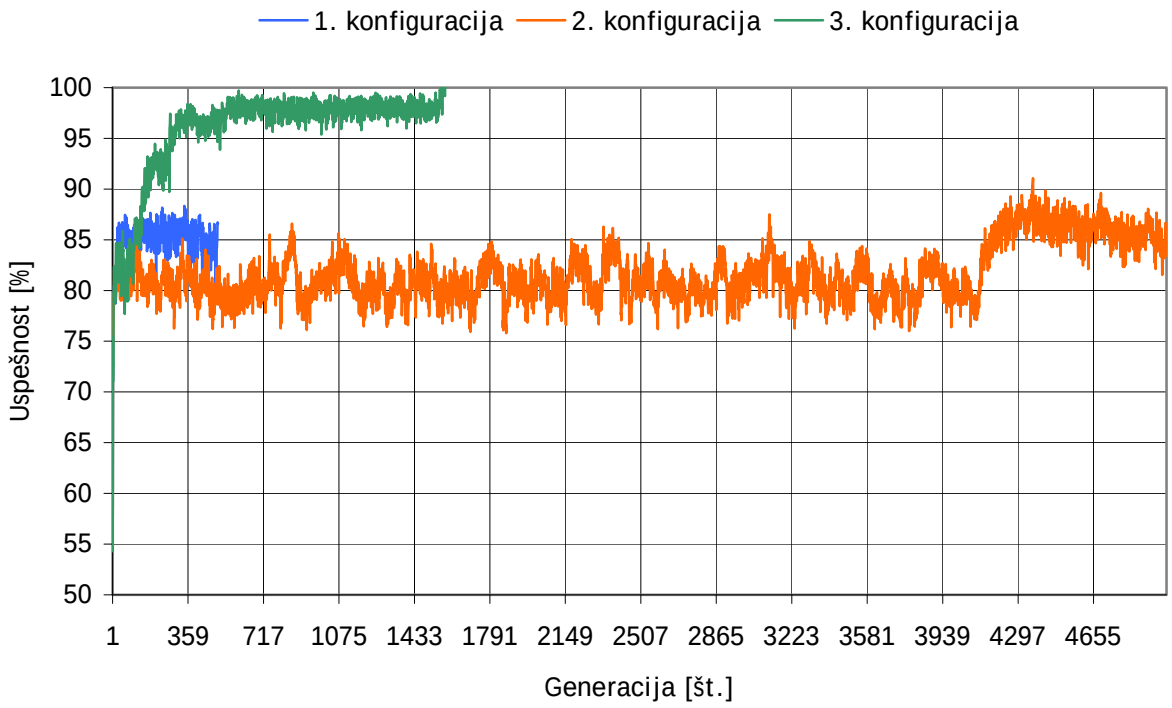
Meritve sem opravil na treh različnih konfiguracijah algoritma. Pri 1. sem pognal **500** generacij, pri 2. pa **5000**. Pri 3. konfiguraciji sem po naključni napolnitvi pravokotnika dodatno nastavil stanja celic v njegovih **kotih na 1** ter pognal algoritem za **5000** generacij.

Vsako konfiguracijo algoritma sem pognal petkrat in za prikaz meritev izbral eno, na oko najprimernejšo.

Konfiguracija algoritma	Uspešnost nabora pravil	Najuspešnejša generacija	Število izvedenih generacij	Trajanje izvajanja
1.	88,34 %	341.	500	14,77 s
2.	91,07 %	4367.	5000	143,41 s
3.	100,00 %	1578.	1578	46,34 s

Tabela 1 - Rezultati meritev.

Uspešnost pravil



Graf 1 - Časovni potek (po generacijah) povprečne uspešnosti nabora pravil za vse tri konfiguracije algoritma. Graf prikazuje uspešnost od 50 % navzgor.

Razvoj pri najboljših pravilih

V spodnjih tabelah so zapisana stanja posameznih celic CA-ja ob začetni (naključni) konfiguraciji (0) in razvoj vseh dvanajstih korakov (1-12) ob najboljšem naboru pravil, ki je nastopil v najboljši generaciji. S sivo barvo je označen prostor zunaj pravokotnika (črne celice so znotraj pravokotnika).

00000000	00000000	00000000	00000000	00000000
00000000	00100000	01010000	01010000	00010000
0111110	0011110	0011110	0101110	0011110
0101001	0001100	0011111	0011111	0101111
0011010	0111101	0111110	0111110	0111110
00000000	00000000	00000010	00000110	00000100
00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000
(0)	**(1)**	**(2)**	**(3)**	**(4)**

00000000	00000000	00000000
00010000	00010000	00010000
0011110	0011110	0011110
0101111	0101111	0101111
0111111	0111111	0111111
00000000	00000010	00000110
00000000	00000000	00000000
00000000	00000000	00000000
(5)	**(6)**	**(7-12)**

Tabela 2 - Časovni razvoj začetne konfiguracije CA-ja pri najboljšem naboru pravil pri 1. konfiguraciji algoritma. Od koraka 7 dalje CA ostane enak. Pravilnost CA-ja je 90,63 %.

00000000	00000000	00000000
00000000	00000000	00000000
0011110	0011110	0011110
0011110	0011110	0011110
0011010	0011010	0011110
0011010	0011110	0011110
00000000	00000100	00001000
00000000	00000000	00000000
(0)	**(1)**	**(2-12)**

Tabela 3 - Časovni razvoj začetne konfiguracije CA-ja pri najboljšem naboru pravil pri 2. konfiguraciji algoritma. Od 2. koraka dalje ostane CA enak. Pravilnost CA-ja je kar 98,44 % - samo ena celica ima napačno stanje.

00000000	00000000
00000000	00000000
0011011	0011111
0001111	0011111
0011011	0011111
00000000	00000000
00000000	00000000
00000000	00000000
(0)	**(1-12)**

Tabela 4 - Časovni razvoj začetne konfiguracije CA-ja pri najboljšem naboru pravil pri 3. konfiguraciji algoritma. Že v 1. koraku CA doseže 100 % pravilnost.

Najpogostejša pravila

Pravila so zaradi svoje velikosti (32 bitna) zapisana v šestnajstiški obliki s pripono "0x".

Pravilnostna tabela	0xFFFFAFF0	0xFFFFDE68	0x80998408
00000	0	0	0
00001	0	0	0

00010	0	0	0
00011	0	1	1
00100	1	0	0
00101	1	1	0
00110	1	1	0
00111	1	0	0
01000	1	0	0
01001	1	1	0
01010	1	1	1
01011	1	1	0
01100	0	1	0
01101	1	0	0
01110	0	1	0
01111	1	1	1
10000	1	1	1
10001	1	1	0
10010	1	1	0
10011	1	1	1
10100	1	1	1
10101	1	1	0
10110	1	1	0
10111	1	1	1
11000	1	1	0
11001	1	1	0
11010	1	1	0
11011	1	1	0
11100	1	1	0
11101	1	1	0
11110	1	1	0
11111	1	1	1

Tabela 5 - Tri najpogostejša pravila v vseh treh konfiguracijah algoritma.

Seznam ostalih zelo pogostih pravil:
0x7F78AAC0
0xE4BE70A2
0x2E75604C
0xFFFFEEEC
0xFFFE97B4
0xFFFFEFF0

Sklep

Meritve so pokazale razmeroma slab rezultat. Uspešnost pravil bi se naj gibala med 90 % pa vse tja do 99 %. Vse to nakazuje, da bi bilo treba algoritem izvajati dlje časa, na precej večjem številu generacij (10-100 krat).
Zanimivo pa je, da je v 3. konfiguraciji algoritma, kjer sem stanja celic v kotih pravokotnika nastavil na 1, uspešnost precej večja. Večkrat (v mojem primeru kar trikrat od petih poskusov) je uspešnost tudi 100 % in to na primeru "samo" 5000 generacij.

Najpogostejša pravila so bolj ali manj enaka. V bistvu popolnoma enakih pravil niti ni tako veliko, saj se večinoma razlikujejo le v enem, morda dveh bitih. V sredini pravokotnika so tako v večini pravila, ki se pričnejo z 0xFFFF.

Ta dokument je skupaj z izvorno kodo algoritma na voljo na moji spletni strani http://wiki.tinelstudio.net/x/CYAN.